Université de Lille

TP 5.07
Spring-Boot Web MVC

R5.A.09 - Frameworks WEB
BUT3 - Info
Philippe Mathieu
2024–2025

Objectifs

Mise en oeuvre de Spring MVC
Exemple de démarrage TRES basique : https://spring.io/quickstart

Exercice 1 : Préambule

Q1. Créer avec via `spring cli` un projet SpringBoot nommé `tp507` avec groupId `fr.but3` et les starters **Web**, **Devtools**, **H2**, **JPA**, **Lombok**

- **web** : support d'exécution web en MVC, avec un serveur tomcat embarqué,
- **devtools** : outil permettant le rechargement automatique et rapide des pages modifées,
- **h2** : base de données, avec notamment une version `inMemory`
- **jpa** : couche de persistence objet/relationnel
- **lombok** : librairie d'annotations pour automatiser getters, setters, constructeurs etc ...

Q2. Créez un fichier `index.html` dans le répertoire `static` avec dedans `<h1>All is OK !</h1>`

Q3. Compilez, exécutez et testez cette archive avec l'url http://localhost:8080

- d'une part via `mvn spring-boot:run`
- d'autre part via `java -jar target/tp507.jar`

Vous constatez que le `jar` contient tout l'environnement pour sa propre exécution. Il contient son propre support d'exécution (ici Tomcat) et s'execute donc sans rien installer !!

Exercice 2 : Un premier contrôleur

Dans Spring MVC, toute requête passe par des controleurs. `@RequestMapping` permet d'indiquer avec quelle url on peut accéder à la page et `@ResponseBody` permet d'indiquer que le retour vers le client web est retourné par le corps de la méthode. Si ce tag n'est pas présent le `return` doit contenir le nom de la vue à appeler. Le nom de la méthode n'a aucune importance.

Q1. Créez dans `src/main/java/fr/but3/tp507` un fichier `ControleurQ1.java` avec une méthode `hello()` associée à un mapping `/hello` renvoyant dans son corps le message "Mon premier controleur"

```
@Controller
class ControleurQ1 {
@RequestMapping("/hello")
@ResponseBody
    public String hello() {
        return "Mon premier controleur";
    }
}
```

Q2. Compilez et testez
Cette fois http://localhost:8080/hello affiche "Mon premier controleur" via la classe `ControleurQ1`

Q3. Il est possible de mettre plusieurs mappings sur la même ressource. `@RequestMapping({"/hello","/"})`. On constate que ce mapping prend l'ascendant sur la page statique `index.html`. On accède maintenant à la méthode du contrôleur

- via l'url http://localhost:8080/hello
- via l'url http://localhost:8080

Exercice 3 : Une première vue en JSP

SpringBoot supporte les serveurs Tomcat, Undertow et Jetty. Par défaut il est fourni avec Tomcat mais sans Jasper. On peut donc faire par défaut des servlets, des filtres ou du Rest, mais pas des JSP. Par défaut, le moteur de templates est thymleaf. Nous allons néanmoins paramétrer Spring pour utiliser des JSP.

Q1. Modifiez le pom en ajoutant Jasper, le compilateur de JSP.

```
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>compile</scope>
</dependency>
```

Q2. Modifiez le fichier `application.properties` et ajoutez le paramétrage des vues en jsp

```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

Q3. Créez dans `src/main` un répertoire `webapp/WEB-INF/jsp`

Q4. Créez une page `mavue.jsp` contenant `<h1>Hello World !</h1>`

Q5. Modifiez le contrôleur pour qu'il appelle cette vue

```
@Controller
class ControleurQ1 {
@RequestMapping("/hello")
    public String hello() {
    return "mavue";
    }
 }
```

Q6. Testez l'url habituelle. Cette fois, on constate que le contrôleur appelle directement la vue.

Vous avez maintenant entre les mains la base d'un modèle MVC en Spring : on passe toujours par un contrôleur, et c'est le contrôleur qui appelle la vue avec la partie utile du modèle.

Remarque Une vue est appelée `template` dans le vocabulaire `SpringBoot`. Dans la configuration par défaut, les vues écrites avec `thymeleaf` doivent être placées dans le répertoire `main/resources/templates`.

Exercice 4 : Passage de paramètres

On souhaite maintenant passer un paramètre `name` à notre url de manière à afficher **hello phil** si `?name=phil` a été passé en paramètre, et toujours `Hello World` si aucun paramètre n'est passé

Q1. La requête HTTP doit donc être passée en paramètre à la méthode du controleur. Pour passer le paramètre à la vue on utilise un `ModelMap` dans lequel on range les informations à passer à la vue.

```
String hello(HttpServletRequest request , ModelMap modelmap) {
String name=request.getParameter("name");
if (name==null || name.isEmpty()) name="world";
modelmap.addAttribute("cle",name);
return "mavue";
```

Q2. La vue peut maintenant être modifiée pour pouvoir lire le paramètre dans la map. Les EL expressions peuvent évidemment être utilisées. Il suffit donc de mettre `Hello ${cle}` dans la vue.

Q3. Testez avec les différentes url

`http://localhost:8080`

`http://localhost:8080/?name=`

`http://localhost:8080/?name=phil`

Exercice 5 : Amélioration avec des annotations

En springBoot, il n'est pas conseillé d'accéder aux objets de l'API Servlet comme on l'a fait précédemment. Dans l'entête de la méthode, plutôt que de passer `HttpServletRequest`, il est possible d'utiliser plus élégamment une annotation `@requestParam` qui permet à Spring de faire automatiquement le lien entre le paramètre et une variable de même nom.

On peut par ailleurs ajouter deux spécifications à cette annotation : `defaultValue` qui affecte automatiquement dans le cas d'un paramètre vide ou absent, et `required` qui permet de spécifier si le paramètre est obligatoire ou non (true par défaut).

Q1. Effectuez cette modification. On notera que le code se simplifie grandement.

```
@Controller
public class ControleurQ1 {

    @RequestMapping("/hello")
    String home(@RequestParam(defaultValue="World", required=false) String name ,
                            ModelMap modelmap)
    {
        modelmap.put("cle",name);
        return "mavue";
    }
}
```

Q2. Testez les différentes valeurs de ces paramètres et regardez les effets sur l'accès à la page.

Exercice 6 : Gestion d'un formulaire

Q1. Créez un Pojo non annoté `Etudiant.java` avec les attributs `Etudiant(id,prenom,nom,age,groupe)`. On ne mettra pas de numéro automatique dans un premier temps.

Q2. Créez un contrôleur `ControleurFormulaire.java` avec en attribut privé une `ArrayList<Etudiant>` (qui fera office de BDD)

Q3. Ajoutez à ce contrôleur deux méthodes dont le mapping est `etudiantForm`

— l'une qui traite du `GET`, permettant d'afficher le formulaire `etudiantForm.jsp`

— l'autre qui traite du `POST`, qui récupère l'objet Etudiant dans ses paramètres (construit par auto-binding), qui ajoute cet étudiant à l'`ArrayList` et qui redirige vers une vue `Lister.jsp`

Q4. Créez la JSP `etudiantForm.jsp` contenant le formulaire HTML, avec lors de sa validation, l'appel au contrôleur en Post

Q5. Créez une JSP `Lister.jsp` permettant de lister de contenu de l'`ArrayList` passée dans le modèle.

Exercice 7 : JPA

Cette fois on supprime l'`ArrayList` du contrôleur et on fait en sorte que l'ensemble des étudiants soit stocké dans une table

Q1. Si ce n'est pas déjà fait, ajoutez au pom les dépendances nécessaires (`starter-data-jpa`, postgres ou h2)

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

Q2. Ajoutez dans `application.properties` les caractéristiques de la base de données et le paramétrage de JPA [1]

Q3. Définir le pojo comme un Entity Bean

Q4. Créez une classe `EtudiantRepository.java` de type `CrudRepository`

Q5. Modifiez le contrôleur en supprimant l'`ArrayList` et en utilisant `EtudiantRepository` à la place.

Q6. Modifiez l'étudiant pour que `Id` soit automatiquement généré. Cela necessite bien sûr de changer légèrement `import.sql` ainsi que le formulaire.

Q7. Testez l'application obtenue. Idéalement la table doit être automatiquement créée au lancement (mode `create`) et le formulaire doit permettre d'entrer directement des étudiants dans cette table.

Exercice 8 : Validation

Q1. Ajoutez le `spring-boot-starter-validation` permettant de contrôler/valider les données d'un objet.

Q2. Vérifiez notamment que l'âge de l'étudiant est compris entre 18 et 25, le nom et prénom ne sont pas nuls et que groupe ne contient qu'une lettre. Si rien n'est validé, on retourne au formulaire.

Q3. Ajouter des messages d'erreur si les contrôles ne sont pas valides.

1. (Une liste de toutes les propriétés utilisables se trouve ici)

Comptez au final combien vous avez de lignes de code ?

- un Bean Etudiant dans lequel tout est fait avec lombok
- un repository d'une ligne
- un contrôleur avec 2 méthodes quasi vides

Au grand maximum 20 lignes, et tout est automatiquement généré, vérifié, persisté, packagé ... et dans une application tout en un , avec les 2 serveurs embarqués .... c'est Magique :-)

Exercice 9 : Si il vous reste du temps ... le compteur revisité !

Q1. Réaliser une application SpringBoot qui affiche la fameuse phrase : `Vous avez visité X fois cette page sur les Y visites au total`